BCA611 Video Oyunları için 3B Grafik

Ders 5
Model Transform,
View Transform,
Projection Transform

Zümra Kavafoğlu
https://zumrakavafoglu.github.io/

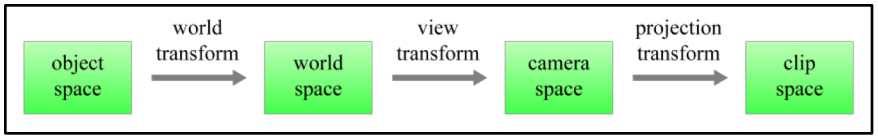
Ders notlarının bu kısmı 3D Computer Graphics for Game Programming(J. Han) kitabının slaytlarından derlenmiştir.

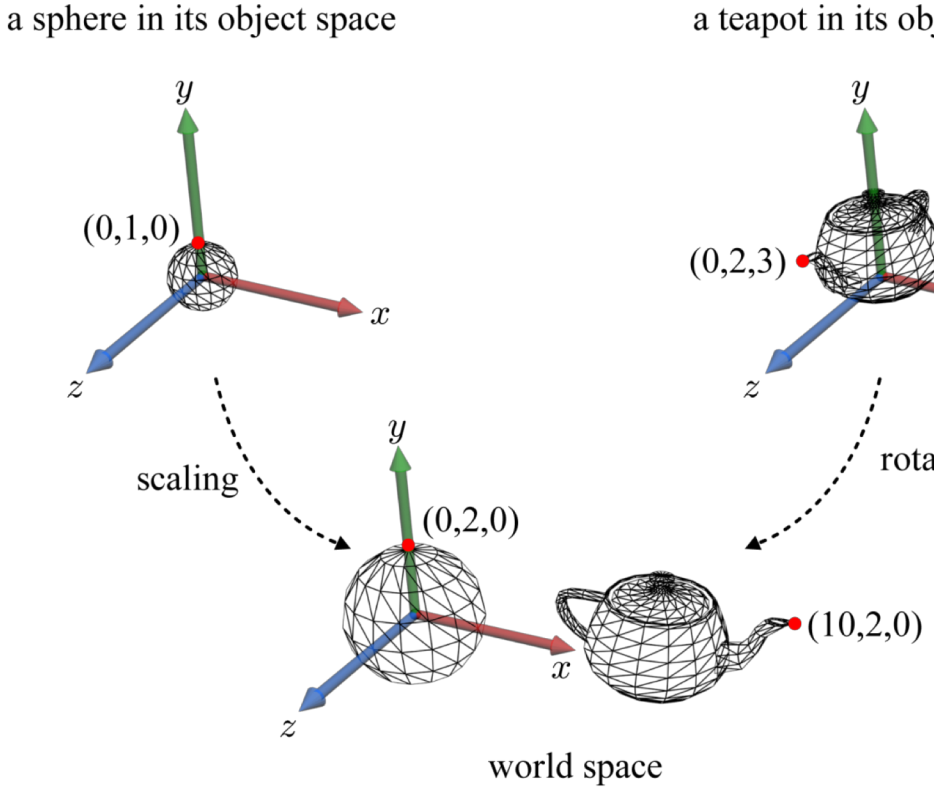
More on Homogeneous Coordinates

- The fourth component of the homogeneous coordinates is not necessarily 1 and is denoted by w. The homogeneous coordinates (x, y, z, w) correspond to the Cartesian coordinates $(x/w, y/w, z/w)$. For example, (1,2,3,1), (2,4,6,2) and (3,6,9,3) are different homogeneous coordinates for the same Cartesian coordinates (1,2,3).
- The w-component of the homogeneous coordinates is used to distinguish between vectors and points.
  - If w is 0, (x, y, z, w) represent a vector.
  - Otherwise, a point.

Model Transform

- A model is created in its own *object space.* The object space for a model typically has no relationship to that of another model.
- The model(*world transform)* 'assembles' all models into a single coordinate system called *world space*.

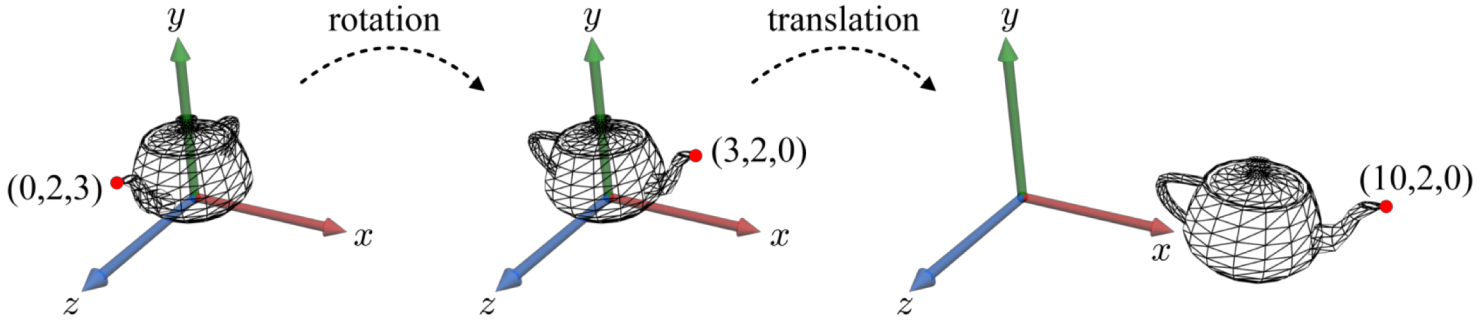




$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

World Transform (cont'd)



$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta_x & -sin\theta_x & 0 \\ 0 & sin\theta_x & cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} cos\theta_y & 0 & sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta_y & 0 & cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} cos\theta_z & -sin\theta_z & 0 & 0 \\ sin\theta_z & cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

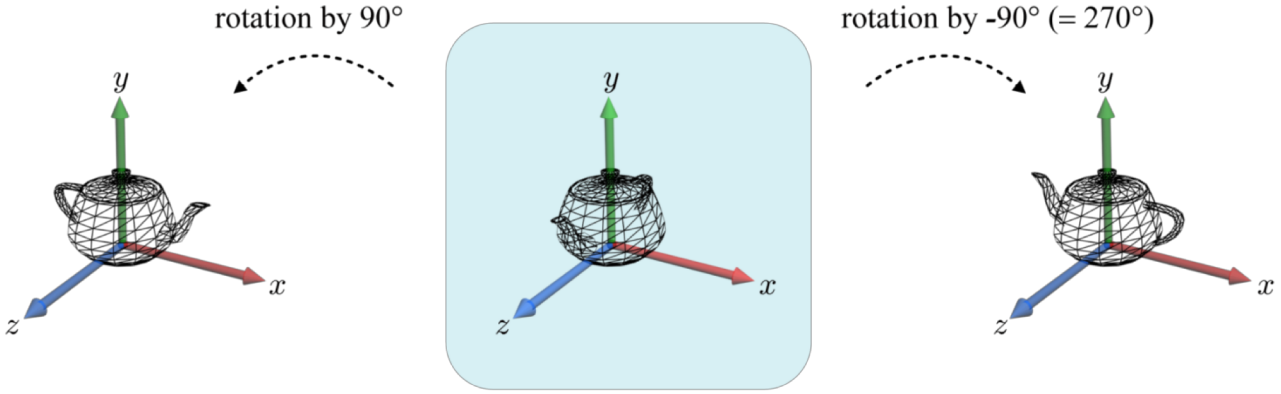
$$R = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 & 0 & 7 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 & 7 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

$$TR = \begin{pmatrix} 1 & 0 & 0 & 7 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 7 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 & 7 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

More on Rotation

- Look at the origin of the coordinate system such that the axis of rotation points toward you. If the rotation is counter-clockwise, the rotation angle is positive. If the rotation is clockwise, it is negative.



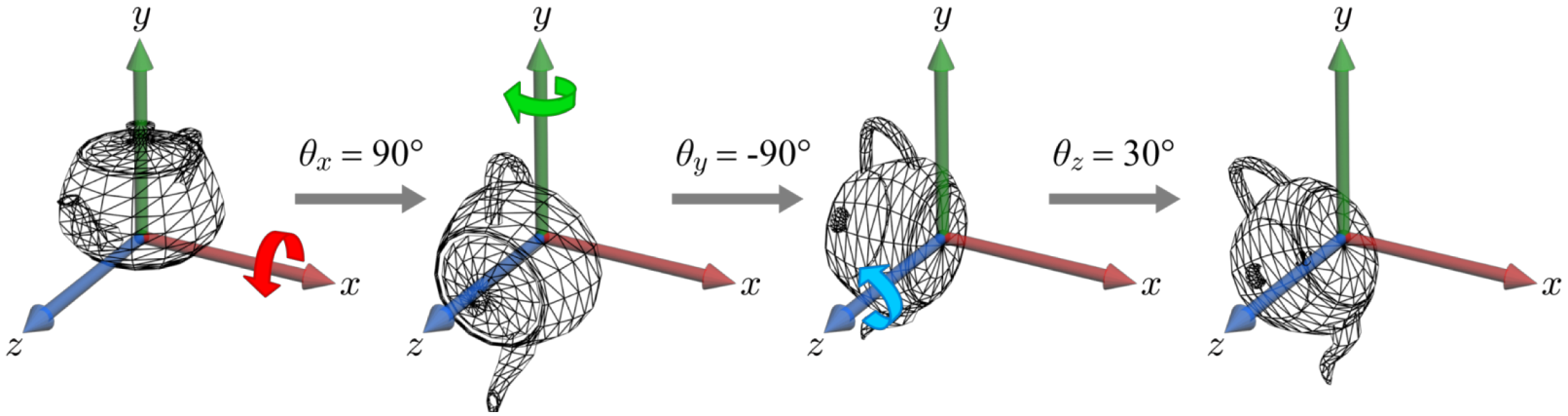
$$R_y(-90°) = \begin{pmatrix} cos(-90°) & 0 & sin(-90°) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(-90°) & 0 & cos(-90°) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} cos\ 270° & 0 & sin\ 270° & 0 \\ 0 & 1 & 0 & 0 \\ -sin\ 270° & 0 & cos\ 270° & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Euler Transform

- When we successively rotate an object about the *x*-, *y*-, and *z*-axes, the object acquires a specific orientation.
- The rotations angles $(\theta_x, \theta_y, \theta_z)$ are called the *Euler angles*. When three rotations are combined into one, it is called Euler transform.



$$R_z(30°)R_y(-90°)R_x(90°) = \begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

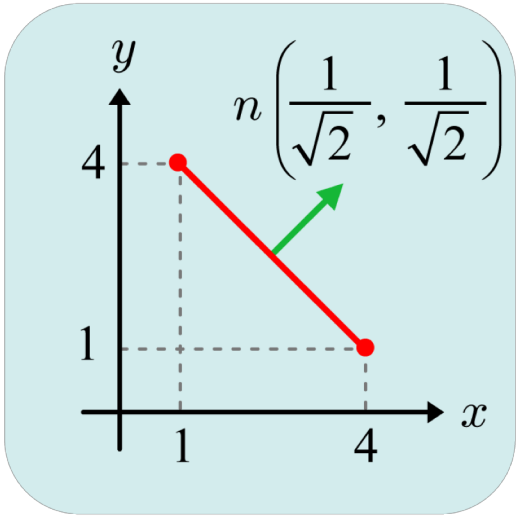
$$= \begin{pmatrix} 0 & -\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Normal Transform

- Let M be the matrix transforming the vertices of a polygon mesh. Can M be used for transforming its normals?

Normal Transform

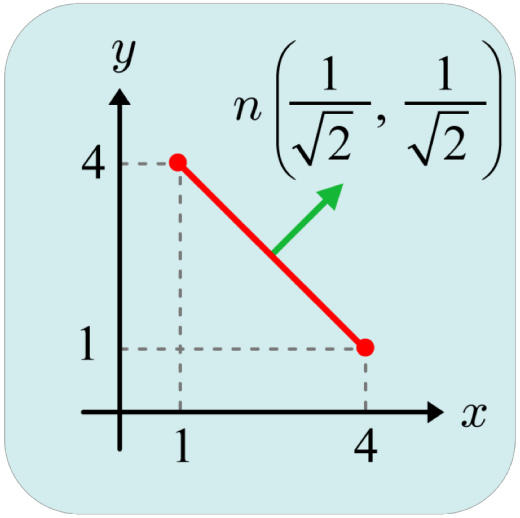
- Let M be the matrix transforming the vertices of a polygon mesh. Can M be used for transforming its normals?



Normal Transform

- Let M be the matrix transforming the vertices of a polygon mesh. Can M be used for transforming its normals?

$$Mn = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{2\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

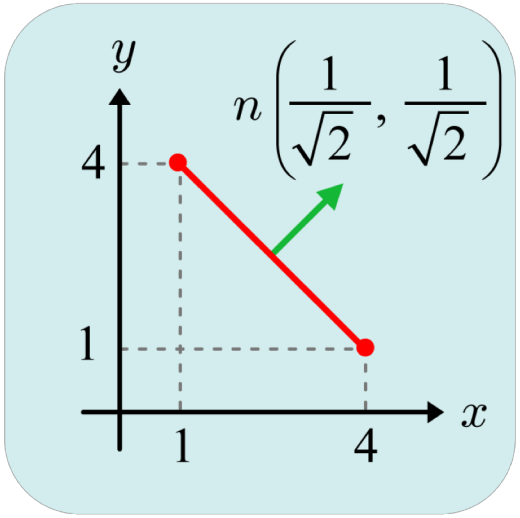


≠

Normal Transform

- Let M be the matrix transforming the vertices of a polygon mesh. Can M be used for transforming its normals?

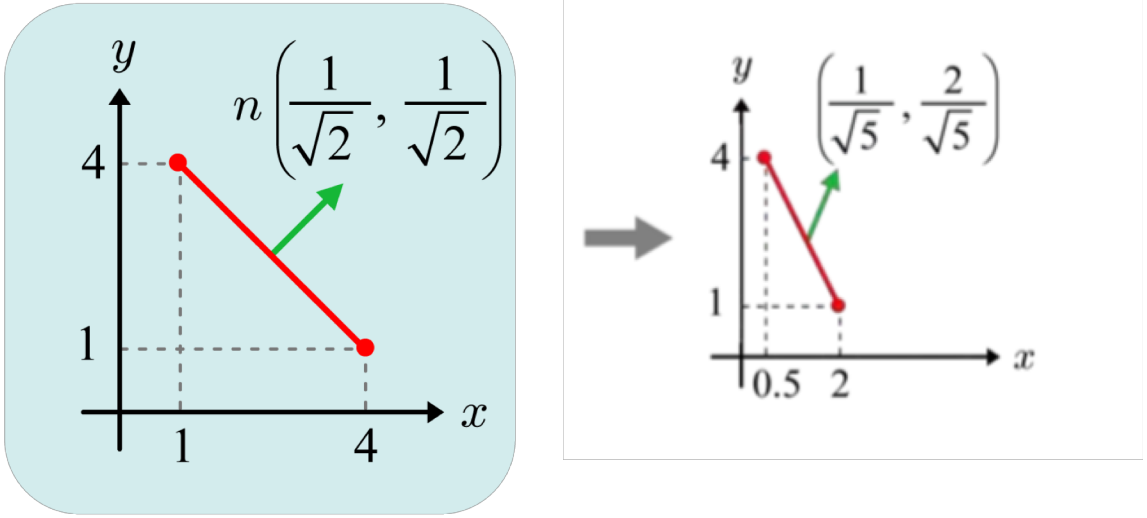
$$Mn = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{2\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \xrightarrow{\text{normalize}} \begin{pmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix}$$



Normal Transform

- Let M be the matrix transforming the vertices of a polygon mesh. Can M be used for transforming its normals?

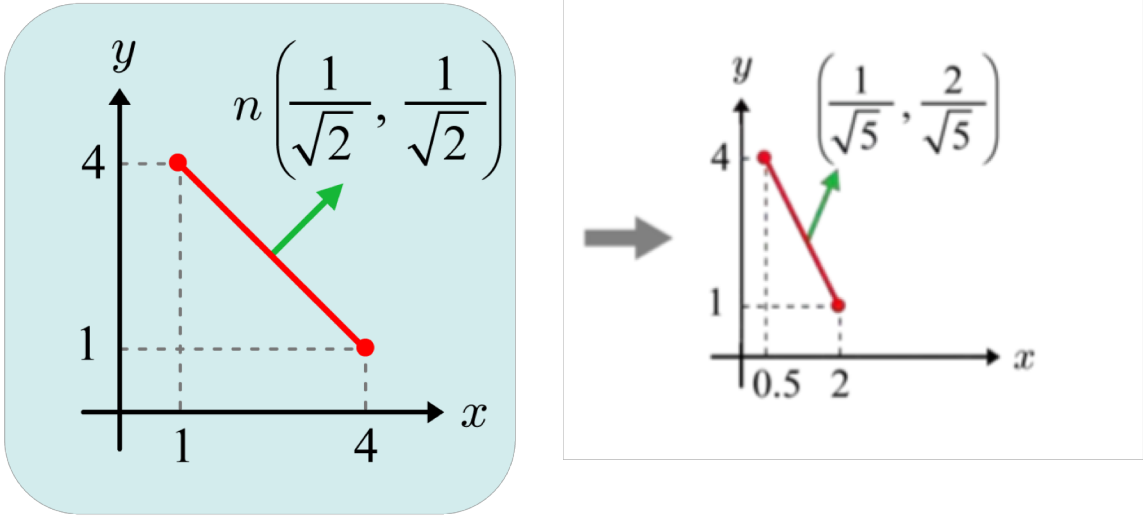
$$Mn = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{2\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \xrightarrow{\text{normalize}} \begin{pmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix}$$



Normal Transform

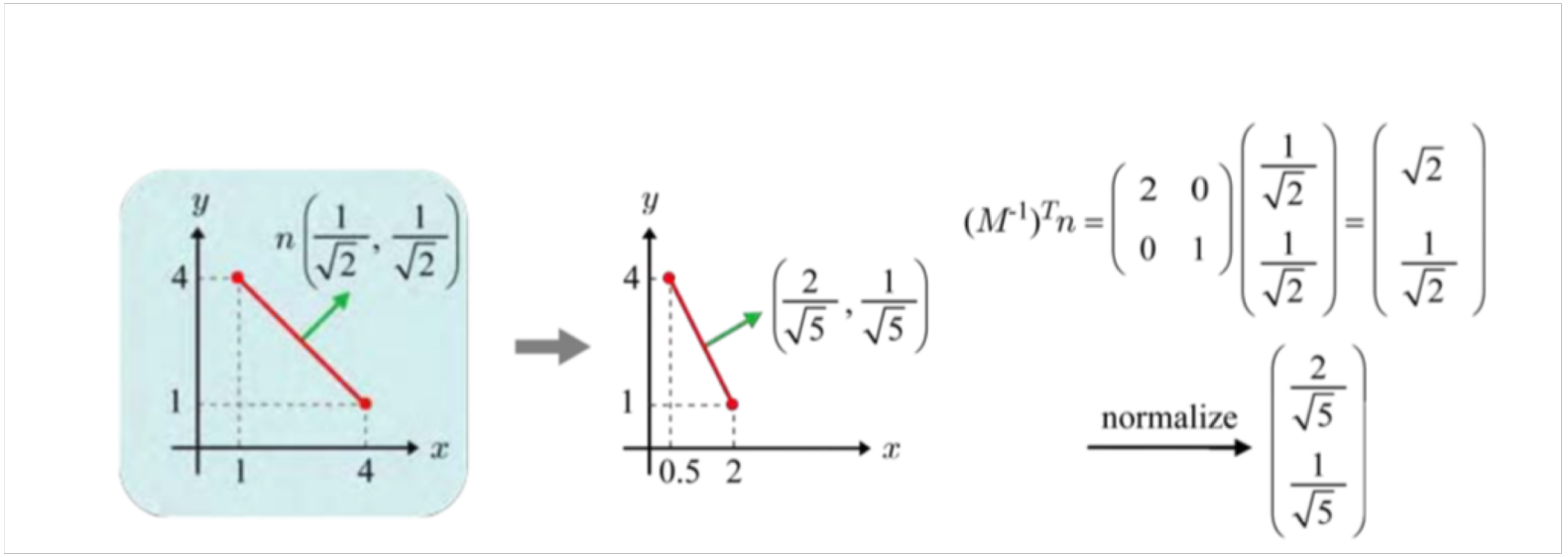
- Let M be the matrix transforming the vertices of a polygon mesh. Can M be used for transforming its normals?
- **If we directly apply M, the answer is NO!**

$$Mn = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{2\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \xrightarrow{\text{normalize}} \begin{pmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix}$$

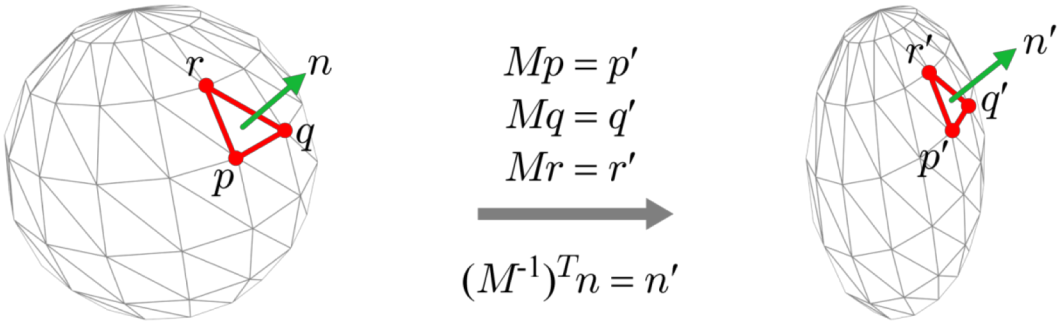


Normal Transform

- Let M be the matrix transforming the vertices of a polygon mesh. Can M be used for transforming its normals?
- **But we can transform normals with $(M^{-1})^T$ if M is a rotation, a translation, a uniform scaling or a combination of those.**



Normal Transform (cont'd)



$$n^T(q-p) = 0$$

$$Mp{=}p'$$

$$Mq{=}q'$$

$$n^T(M^{-1}q' - M^{-1}p') = 0$$

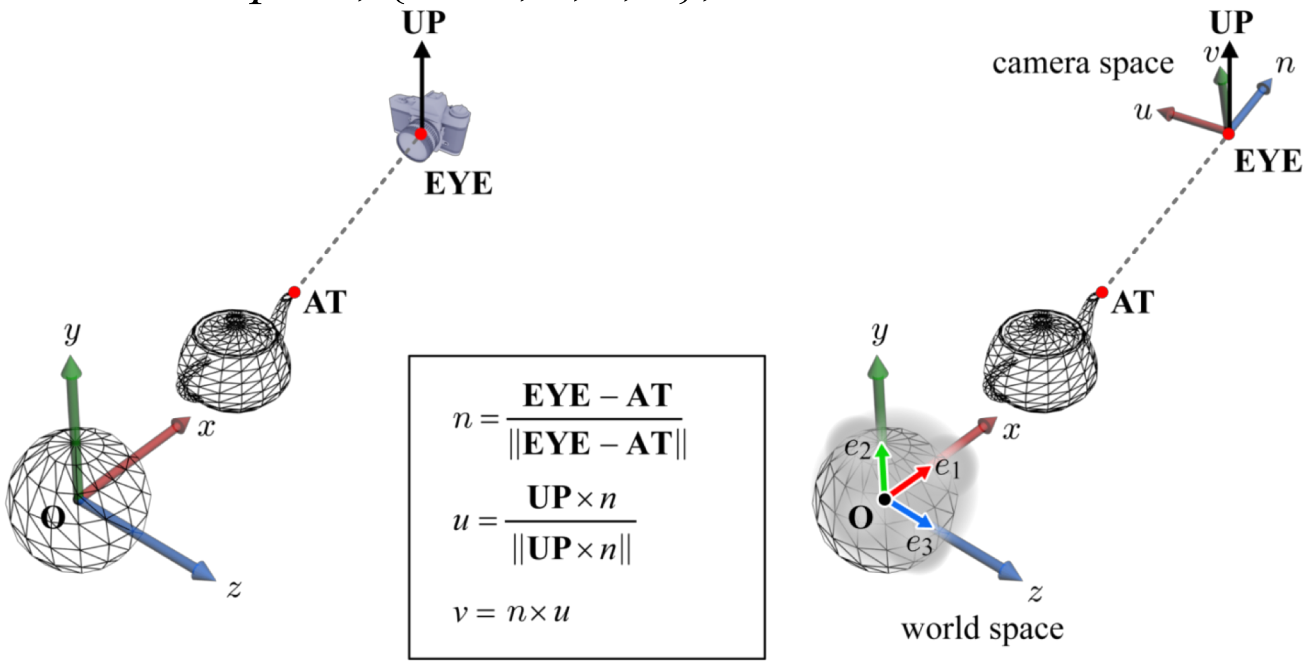
$$n^T M^{-1}(q'-p') = 0$$

$$(q'-p')^T (M^{-1})^T n = 0$$

$$(r'-p')^T (M^{-1})^T n = 0$$

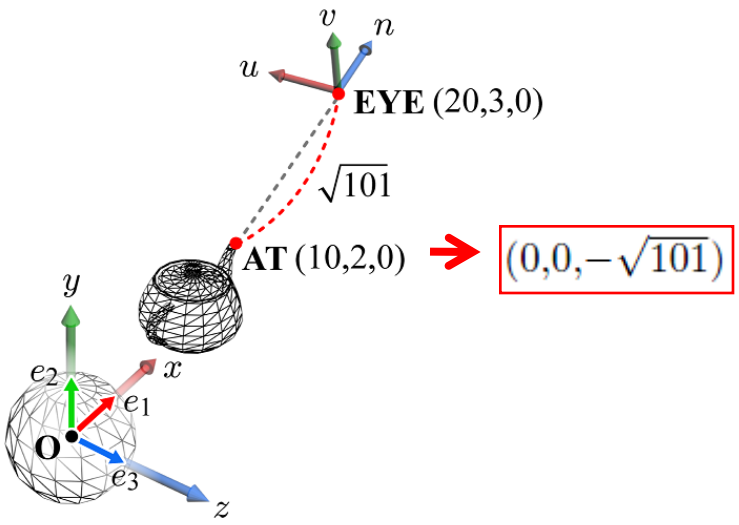
View Transform

- The world transform has been done, and now all objects are in the world space.
- Next step: camera pose specification
  - **EYE**: camera position
  - **AT**: a reference point toward which the camera is aimed
  - **UP**: view up vector that roughly describes where the top of the camera is pointing. (In most cases, **UP** is set to the *y*-axis of the world space.)
- Then, the *camera space*, {**EYE**, u, v, n}, can be created.

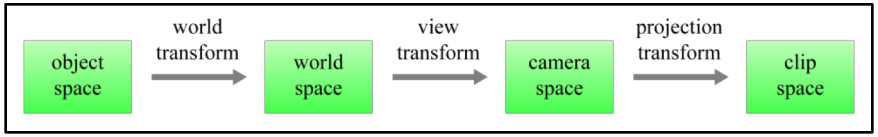


View Transform (cont'd)

- A point is given different coordinates in distinct spaces.

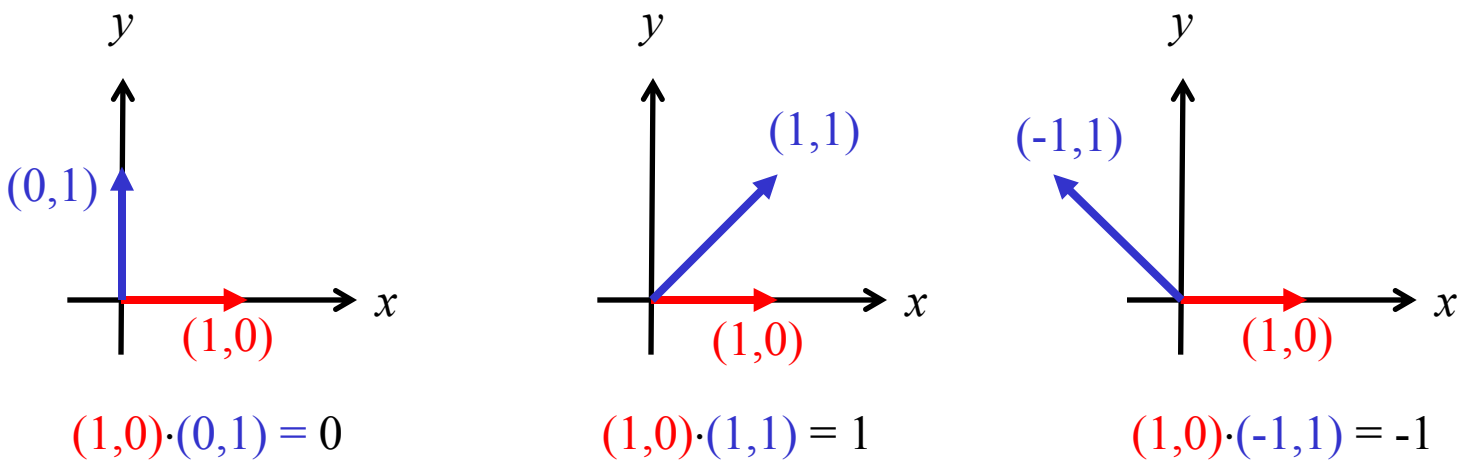


- If all the world-space objects can be newly defined in terms of the camera space in the manner of the teapot's mouth, it becomes much easier to develop the rendering algorithms. In general, it is called *space change*.
- The view transform converts each vertex from the world space to the camera space.



Dot Product

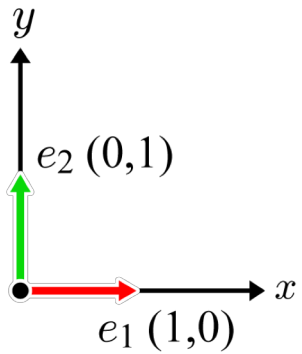
- Given vectors, a and b, whose coordinates are $(a_1, a_2, .. , a_n)$ and $(b_1, b_2, .. , b_n)$, respectively, the dot product $a \cdot b$ is defined to be $a_1b_1+a_2b_2+ .. +a_nb_n$.
- In Euclidean geometry, $a \cdot b = \|a\|\|b\|\cos\theta$, where $\|a\|$ and $\|b\|$ denote the lengths of a and b, respectively, and θ is the angle between a and b.
  - If a and b are perpendicular to each other, $a \cdot b = 0$.
  - If θ is an acute angle, $a \cdot b > 0$.
  - If θ is an obtuse angle, $a \cdot b < 0$.



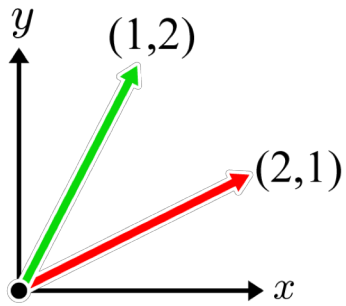
- If a is a unit vector, $a \cdot a = 1$.

Orthonormal Basis

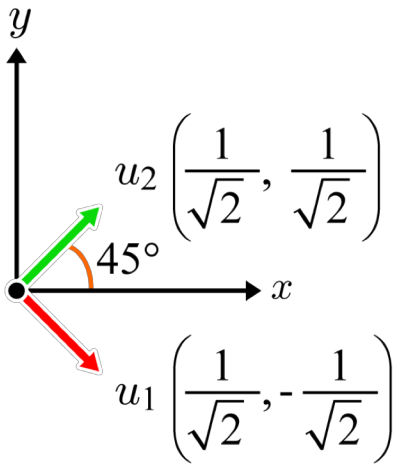
- Orthonormal basis = an orthogonal set of unit vectors



orthonormal
standard

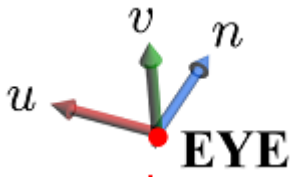


non-orthonormal
non-standard



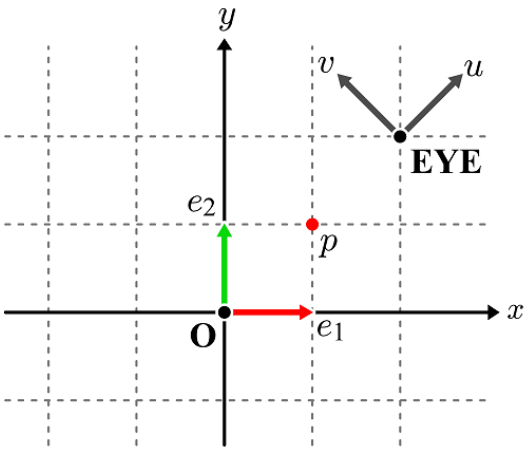
orthonormal
non-standard

- The camera space has an orthonormal basis $\{u, v, n\}$.
- Note that $u \cdot u = v \cdot v = n \cdot n = 1$ and $u \cdot v = v \cdot n = n \cdot u = 0$.



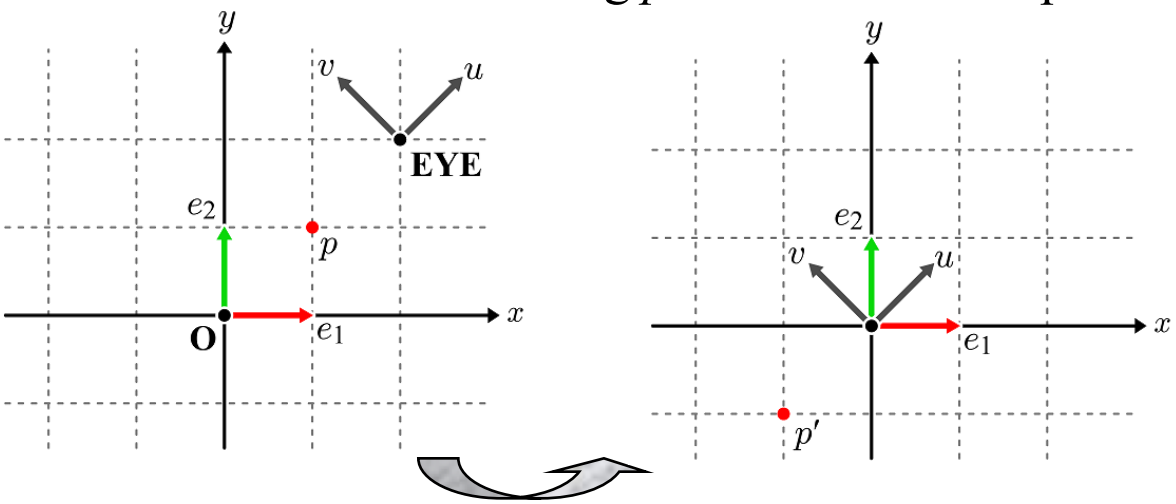
2D Analogy for View Transform

- The coordinates of p are (1,1) in the world space but (-√2,0) in the camera space.
- Let's superimpose the camera space to the world space while maintaining the relative poses of the scene objects with respect to the camera space. Imagining invisible rods connecting p and the camera space such that the transform is



2D Analogy for View Transform

- The coordinates of p are (1,1) in the world space but (-√2,0) in the camera space.
- Let's superimpose the camera space to the world space while maintaining the relative poses of the scene objects with respect to the camera space. Imagining invisible rods connecting p and the camera space such that the transform is

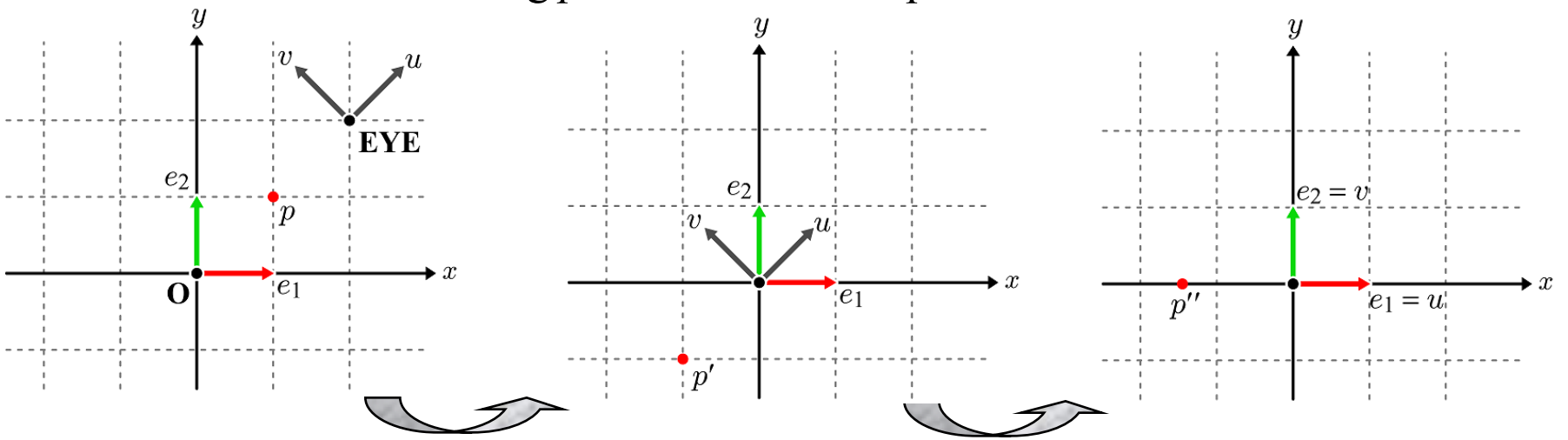


translation by (-2,-2)

$$T = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

2D Analogy for View Transform

- The coordinates of p are (1,1) in the world space but (-√2,0) in the camera space.
- Let's superimpose the camera space to the world space while maintaining the relative poses of the scene objects with respect to the camera space. Imagining invisible rods connecting p and the camera space such that the transform is



translation by (-2,-2)

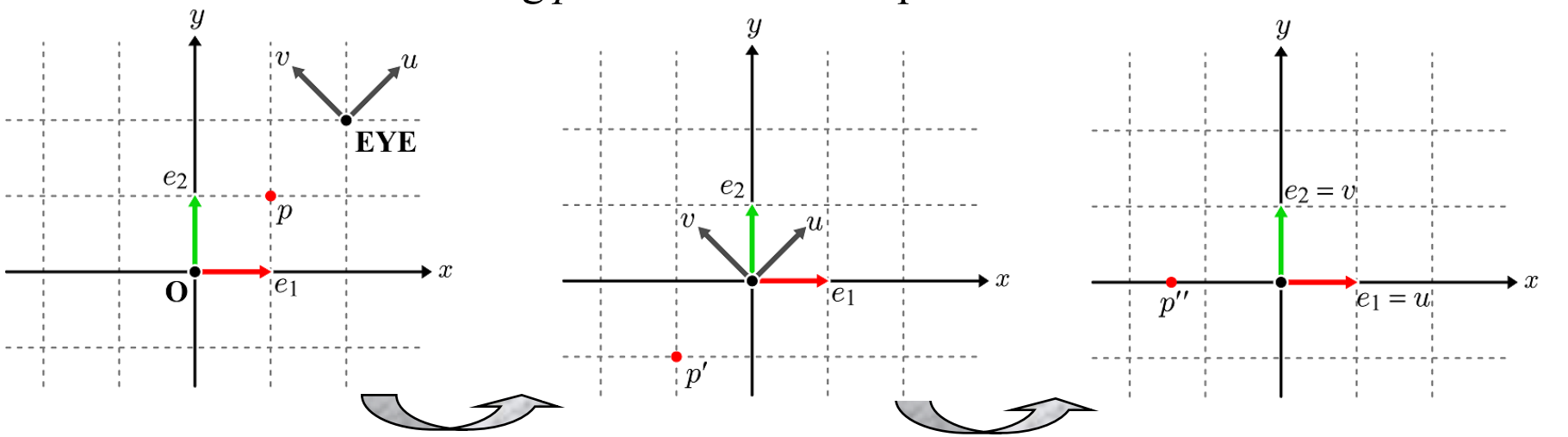
$$T = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

rotation by -45°

$$R = \begin{pmatrix} cos(-45°) & -sin(-45°) & 0 \\ sin(-45°) & cos(-45°) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2D Analogy for View Transform

- The coordinates of p are (1,1) in the world space but (-√2,0) in the camera space.
- Let's superimpose the camera space to the world space while maintaining the relative poses of the scene objects with respect to the camera space. Imagining invisible rods connecting p and the camera space such that the transform is



translation by (-2,-2)

$$T = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

rotation by -45°

$$R = \begin{pmatrix} cos(-45°) & -sin(-45°) & 0 \\ sin(-45°) & cos(-45°) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- As the camera space becomes identical to the world space, the world-space coordinates of p'' can be taken as the camera-space coordinates.

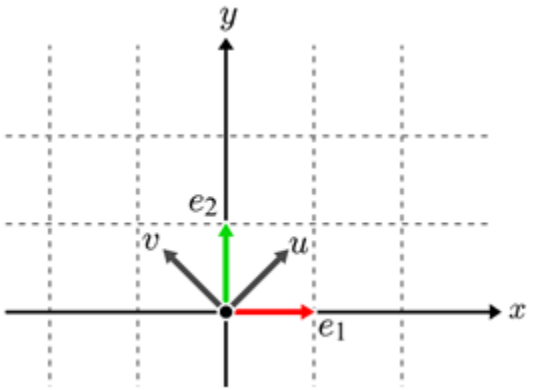
2D Analogy for View Transform (cont'd)

- Let's see if the combination of T and R correctly transforms p.

$$RT = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -2\sqrt{2} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$p'' = RTp = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -2\sqrt{2} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -\sqrt{2} \\ 0 \\ 1 \end{pmatrix}$$

- How to compute R? It is obtained using the camera-space basis vectors.



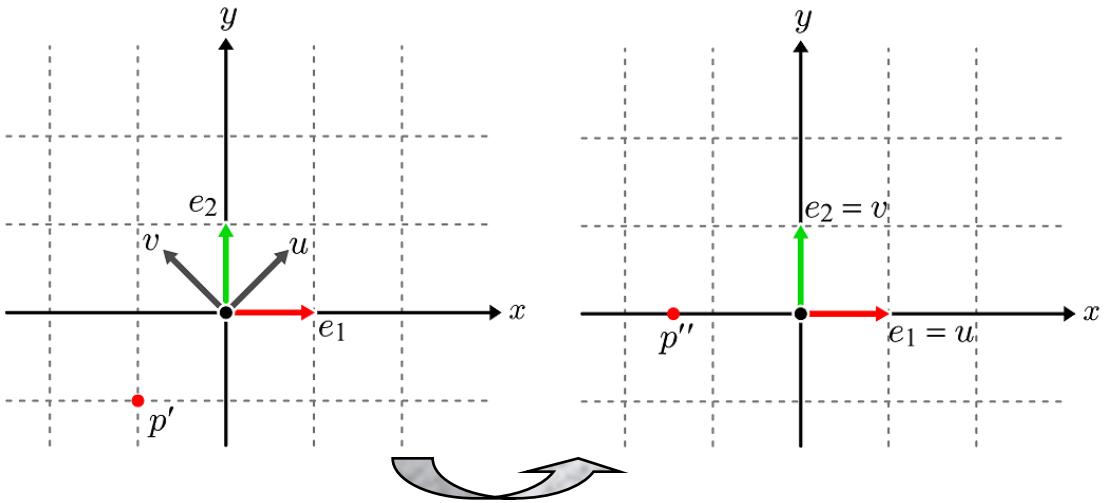
$$R = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

u

v

2D Analogy for View Transform (cont'd)

- It is straightforward to show that R transforms u into e_1 and v into e_2.



$$\begin{matrix} u \\ v \end{matrix}\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \overset{u}{\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}} = \overset{e_1}{\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}}$$

$$\begin{matrix} u \\ v \end{matrix}\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \overset{v}{\begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}} = \overset{e_2}{\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}}$$

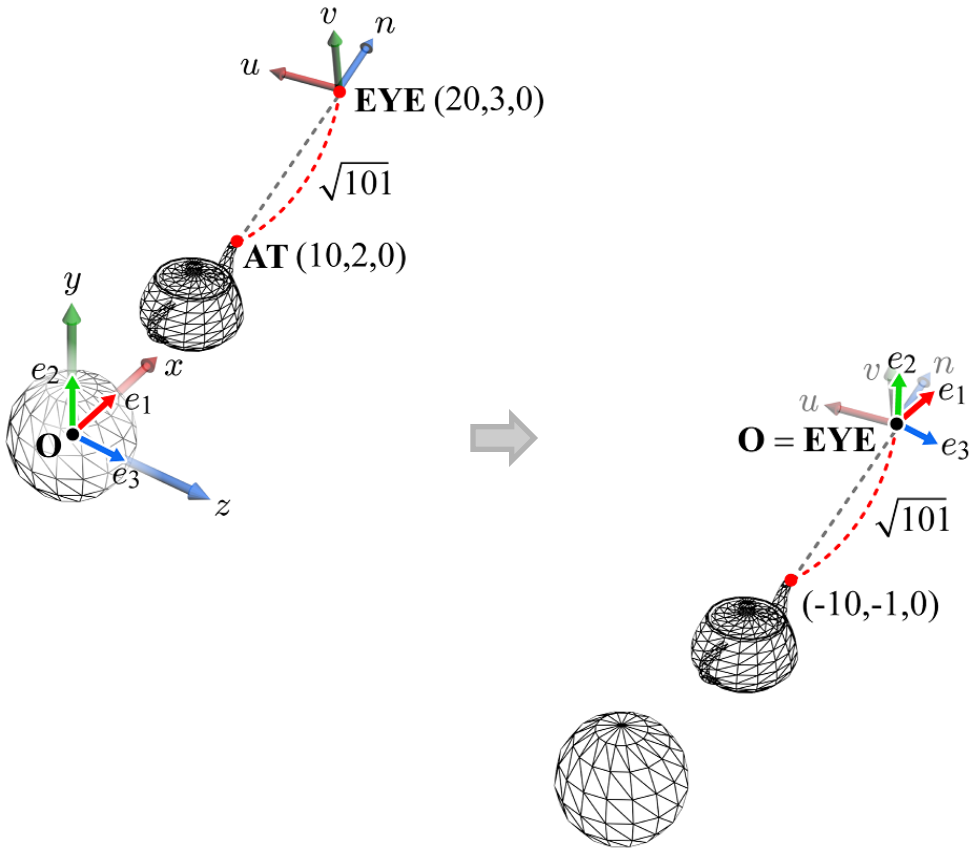
- R converts the coordinates defined in terms of the basis $\{e_1, e_2\}$, e.g., (-1,-1), into those defined in terms of the basis $\{u, v\}$, e.g., $(-\sqrt{2},0)$. In other words, R performs the *basis change* from $\{e_1, e_2\}$ to $\{u, v\}$.

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} -\sqrt{2} \\ 0 \\ 0 \end{pmatrix}$$

- The problem of space change is decomposed into translation and basis change.

View Transform (cont'd)

- Let us do the same thing in 3D. First of all, **EYE** is translated to the origin of the world space. Imagine invisible rods connecting the scene objects and the camera space. The translation is applied to the scene objects.



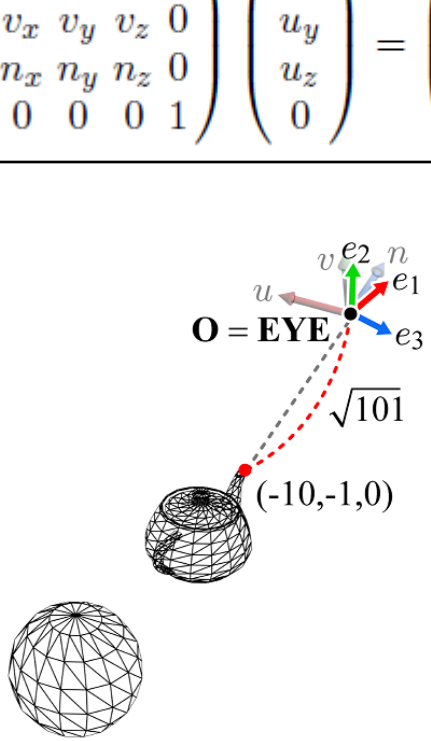
$$\begin{pmatrix} 1 & 0 & 0 & -\mathbf{EYE}_x \\ 0 & 1 & 0 & -\mathbf{EYE}_y \\ 0 & 0 & 1 & -\mathbf{EYE}_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

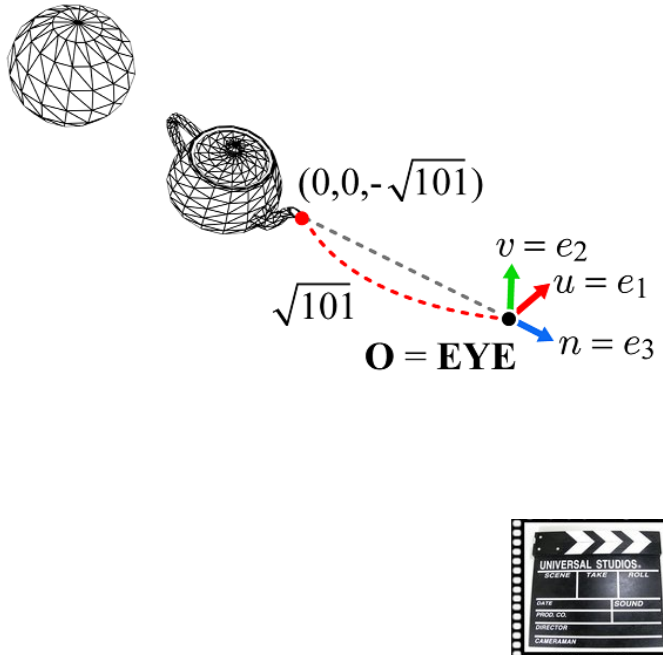
$$\begin{pmatrix} 1 & 0 & 0 & -20 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 \\ 2 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -10 \\ -1 \\ 0 \\ 1 \end{pmatrix}$$

View Transform (cont'd)

- The world space and the camera space now share the origin, due to translation.
- We then need a rotation that transforms u, v, and n into e_1, e_2, and e_3, respectively, i.e., $Ru=e_1$, $Rv=e_2$, and $Rn=e_3$. R performs the *basis change* from $\{e_1, e_2, e_3\}$ to $\{u, v, n\}$.

$$Ru = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = e_1$$





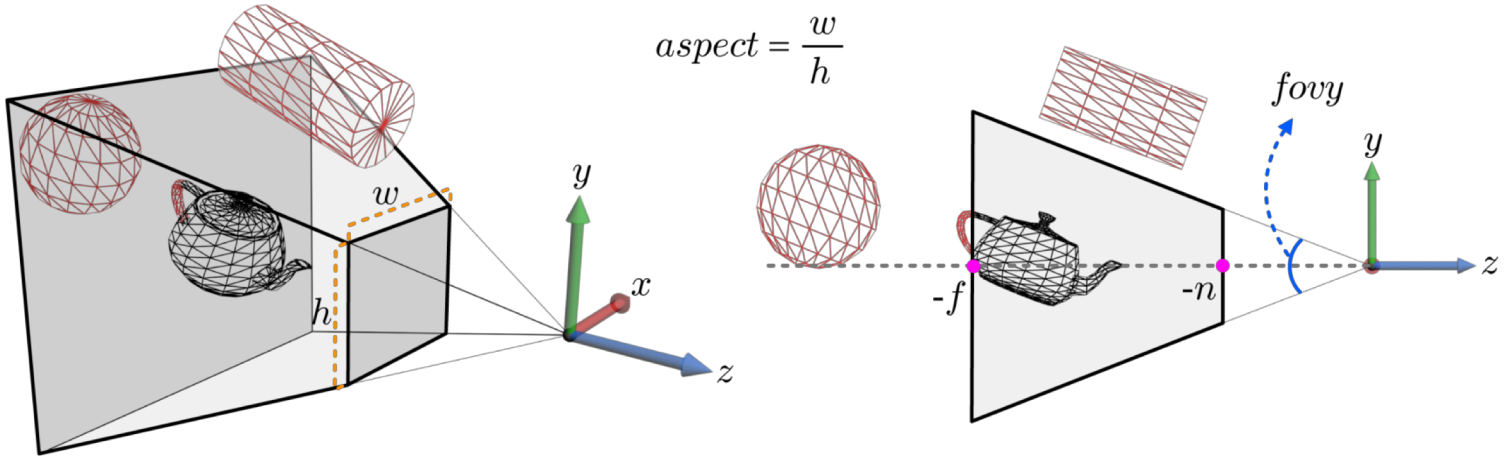
View Transform (cont'd)

- The view matrix

$$\begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\mathbf{EYE}_x \\ 0 & 1 & 0 & -\mathbf{EYE}_y \\ 0 & 0 & 1 & -\mathbf{EYE}_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z & -\mathbf{EYE} \cdot u \\ v_x & v_y & v_z & -\mathbf{EYE} \cdot v \\ n_x & n_y & n_z & -\mathbf{EYE} \cdot n \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

View Frustum

- Let us denote the basis of the camera space by $\{x, y, z\}$ instead of $\{u, v, n\}$.
- Recall that, for constructing the view transform, we defined the external parameters of the camera, i.e., **EYE**, **AT**, and **UP**. Now let us control the camera's internals. It is analogous to choosing a lens for the camera and controlling zoom-in/zoom-out.
- The *view frustum* parameters, *fovy*, *aspect*, *n*, and *f*, define a truncated pyramid.

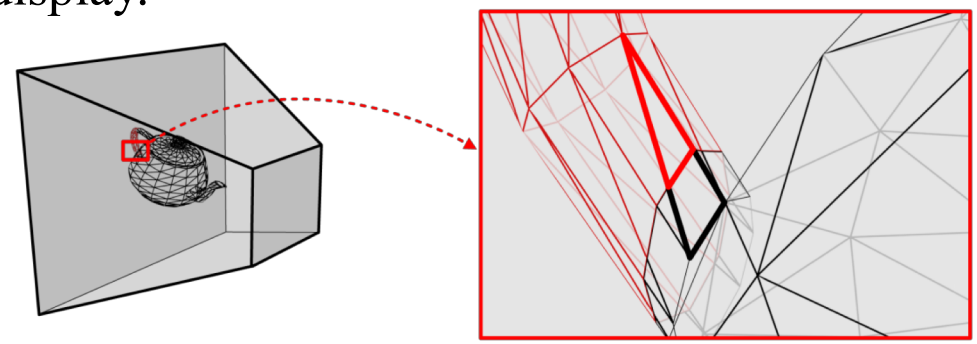


- The near and far planes run counter to the real-world camera or human vision system, but have been introduced for the sake of computational efficiency.

View Frustum (cont'd)

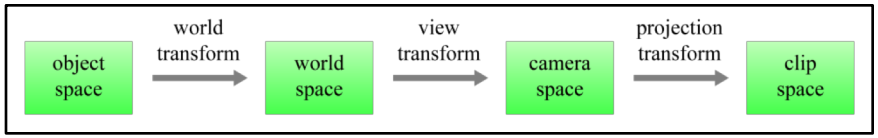
- View-frustum culling
  - A large enough box or sphere bounding a polygon mesh is computed at the preprocessing step, and then at run time a CPU program tests if the bounding volume is outside the view frustum. If it is, the polygon mesh is discarded and does not enter the rendering pipeline.
  - It can save a fair amount of GPU computing cost with a little CPU overhead.
- The cylinder and sphere would be discarded by the view-frustum culling whereas the teapot would survive.

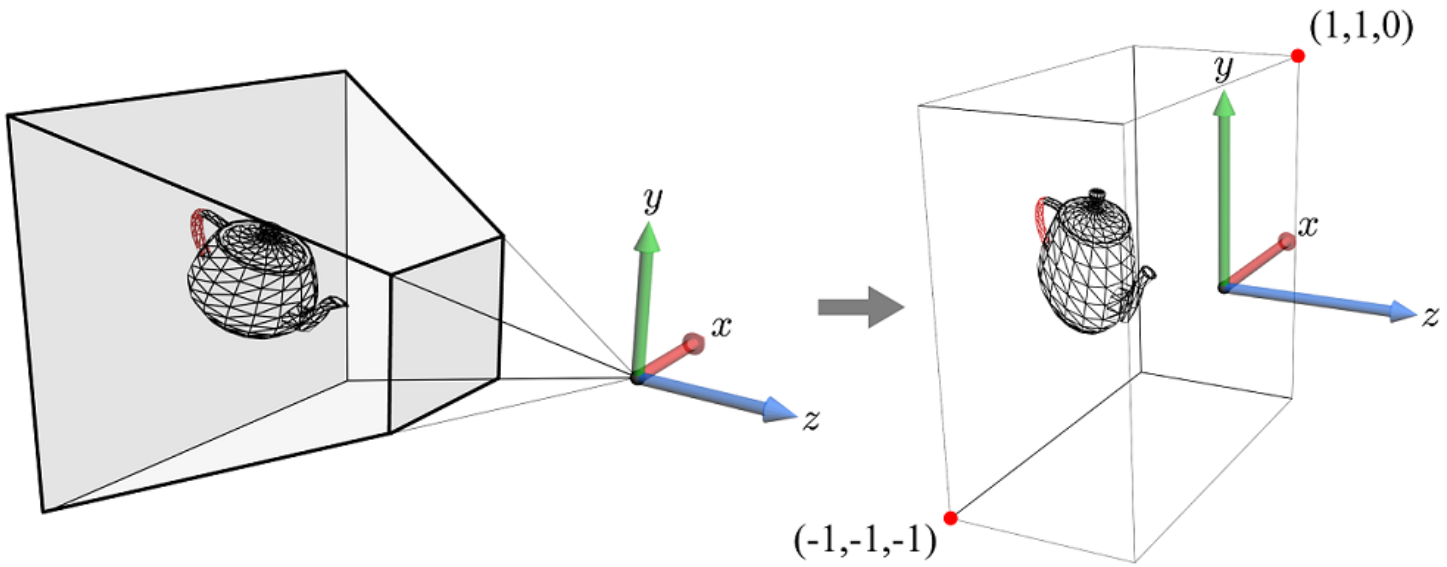
- If a polygon intersects the boundary of the view frustum, it is *clipped* with respect to the boundary, and only the portion inside the view frustum is processed for display.



Projection Transform

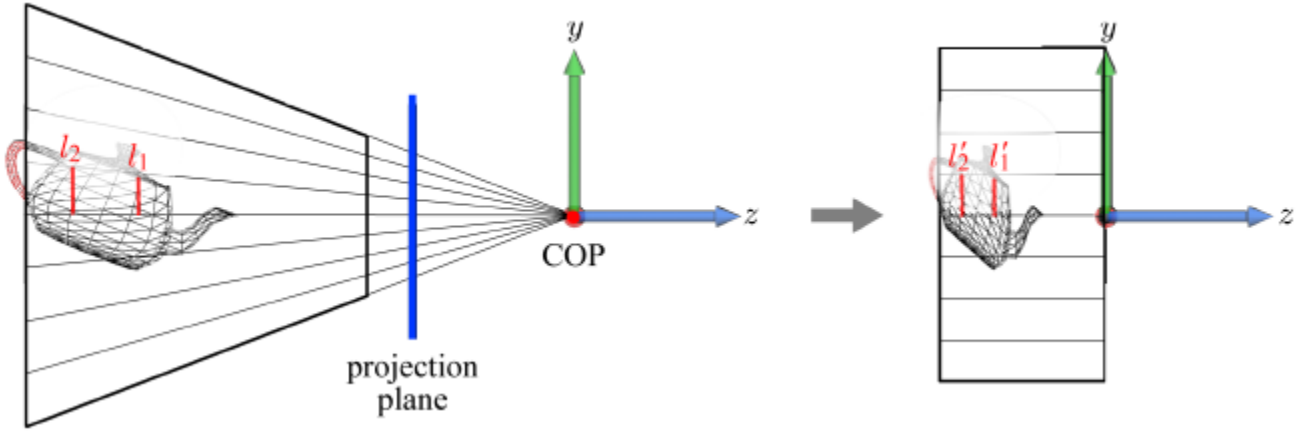
- It is not easy to clip the polygons with respect to the view frustum.
- If there is a transform that converts the view frustum to the axis-aligned box, and the transform is applied to the polygons of the scene, clipping the transformed polygons with respect to the box is much easier.
- The transform is called *projection transform*.





Projection Transform (cont'd)

- The view frustum can be taken as a convergent pencil of projection lines. The lines converge on the origin, where the camera (**EYE**) is located. The origin is often called the center of projection (COP).
- All 3D points on a projection line are mapped onto a single 2D point in the projected image. It brings the effect of perspective projection, where objects farther away look smaller.



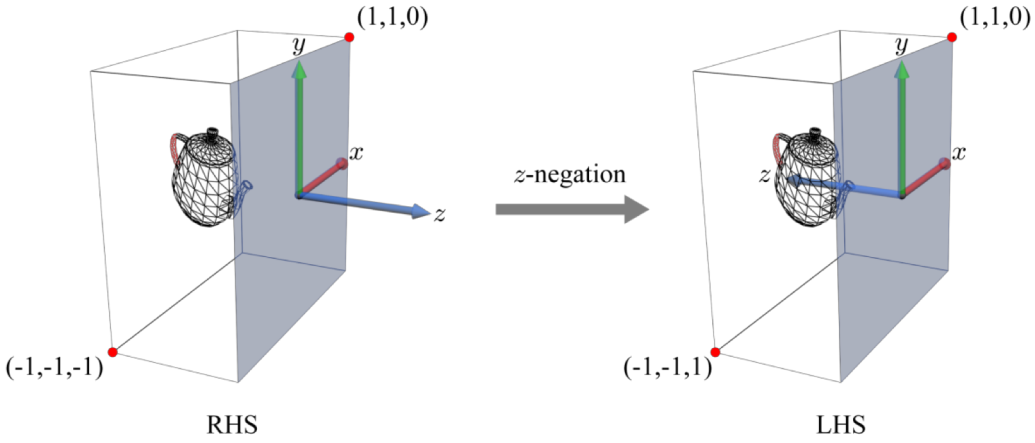
- The projection transform ensures that the projection lines become parallel, i.e., we have a *universal* projection line. Now viewing is done along the universal projection line. It is called the *orthographic projection*. The projection transform brings the effect of perspective projection "within a 3D space."

Projection Transform (cont'd)

- Projection transform matrix

$$\begin{pmatrix} \frac{cot(\frac{fovy}{2})}{aspect} & 0 & 0 & 0 \\ 0 & cot(\frac{fovy}{2}) & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

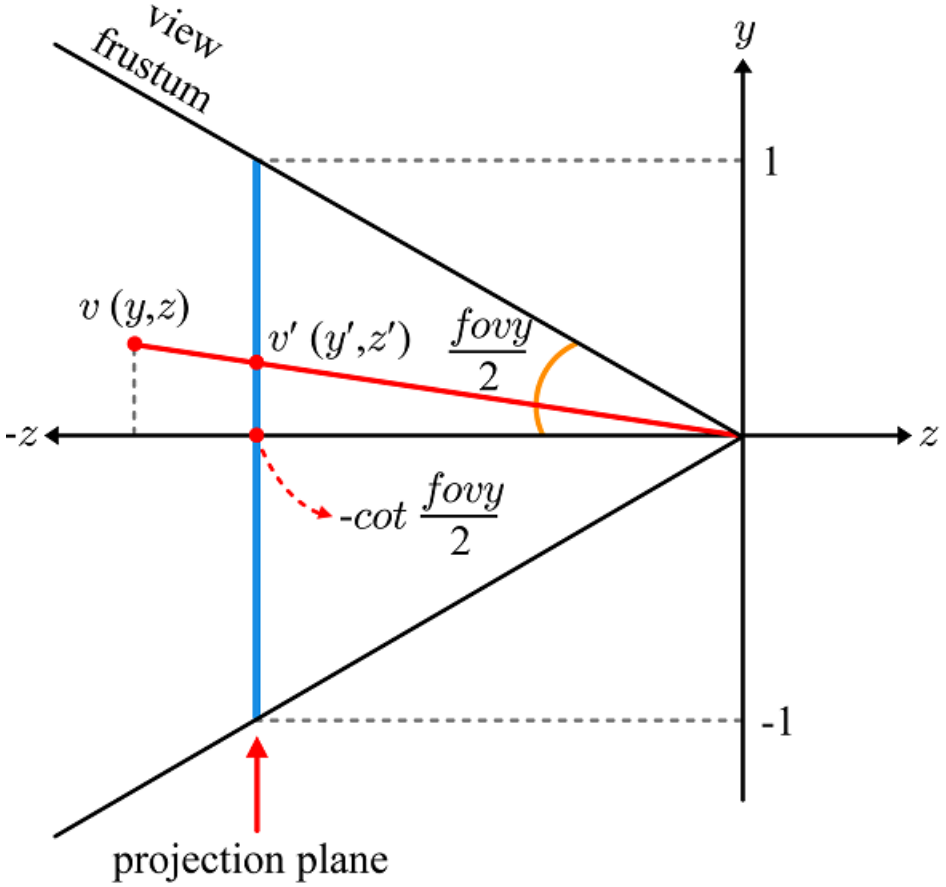
- The projection-transformed objects will enter the rasterization stage.
- Unlike the vertex processing stage, the rasterization stage is implemented in hardware, and assumes that the clip space is left-handed. In order for the vertex processing stage to be compatible with the hard-wired rasterization stage, the objects should be *z*-negated.



$$\begin{pmatrix} \frac{cot(\frac{fovy}{2})}{aspect} & 0 & 0 & 0 \\ 0 & cot(\frac{fovy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{f}{f-n} & -\frac{nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Deriving Projection Transform

- Based on the fact that projection-transformed y coordinate (y') is in the range of [-1,1], we can compute the general representation of y'.



$$y : z = y' : -\cot\frac{fovy}{2}$$

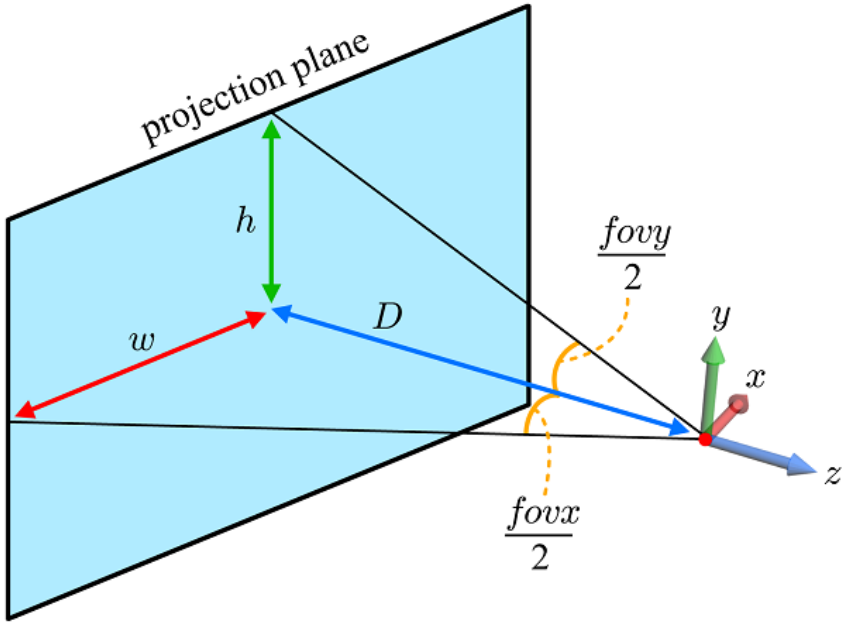
$$\rightarrow \quad y' = -\cot\frac{fovy}{2} \cdot \frac{y}{z}$$

$$x' = -\cot\frac{fovx}{2} \cdot \frac{x}{z}$$

- As shown above, we could compute x' in a similar way if *fovx* were given.

Deriving Projection Transform (cont'd)

- Unfortunately *fovx* is not given, and therefore let's define x' in terms of *fovy* and *aspect*.



$$x' = -\cot\frac{fovx}{2}\cdot\frac{x}{z}$$

$$\downarrow$$

$$x' = -\cot\frac{fovx}{2}\cdot\frac{x}{z} = -\frac{\cot\frac{fovy}{2}}{aspect}\cdot\frac{x}{z}$$

$$\uparrow$$

$$\cot\frac{fovx}{2} = \frac{\cot\frac{fovy}{2}}{aspect}$$

$$\frac{w}{D} = \tan\frac{fovx}{2} \rightarrow w = D\cdot\tan\frac{fovx}{2}$$

$$\frac{h}{D} = \tan\frac{fovy}{2} \rightarrow h = D\cdot\tan\frac{fovy}{2}$$

$$\rightarrow \frac{w}{h} = \frac{\tan\frac{fovx}{2}}{\tan\frac{fovy}{2}} = \frac{\cot\frac{fovy}{2}}{\cot\frac{fovx}{2}}$$

$$\uparrow$$

Deriving Projection Transform (cont'd)

- We have found x' and y'.

$$x' = -\cot\frac{fovx}{2}\cdot\frac{x}{z} = -\frac{\cot\frac{fovy}{2}}{aspect}\cdot\frac{x}{z}$$

D

A

$$y' = -\cot\frac{fovy}{2}\cdot\frac{y}{z}$$

D

- Homogeneous coordinates representation

$$v' = (x', y', z', 1) = (-\frac{D}{A}\cdot\frac{x}{z}, -D\frac{y}{z}, z', 1) \rightarrow (\frac{D}{A}x, Dy, -zz', -z)$$

- Then, we have the following projection matrix. Note that z' and z'' are independent of x and y, and therefore m_1 and m_2 are 0s.

$$\begin{pmatrix} \frac{D}{A}x \\ Dy \\ z'' \\ -z \end{pmatrix} = \begin{pmatrix} \frac{D}{A} & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ m_1 & m_2 & m_3 & m_4 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

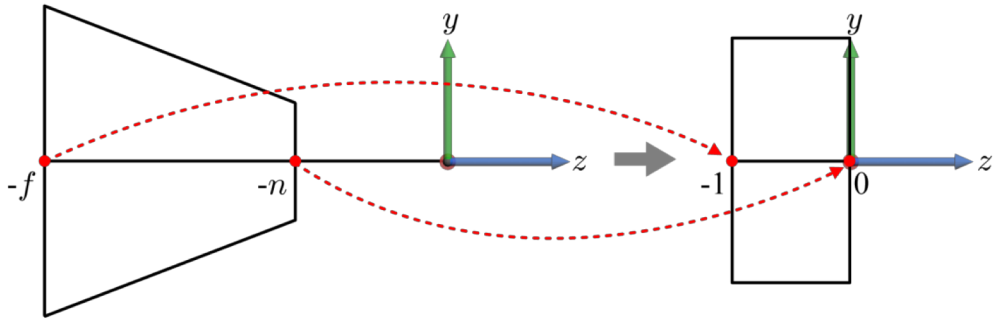
just 0s

Deriving Projection Transform (cont'd)

- Let's apply the projection matrix.

$$\begin{pmatrix} \frac{D}{A} & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ 0 & 0 & m_3 & m_4 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{D}{A}x \\ Dy \\ m_3 z + m_4 \\ -z \end{pmatrix} \rightarrow \begin{pmatrix} -\frac{D}{A} \cdot \frac{x}{z} \\ -D\frac{y}{z} \\ -m_3 - \frac{m_4}{z} \\ 1 \end{pmatrix} = v'$$

- In projection transform, observe that $-f$ and $-n$ are transformed to -1 and 0, respectively.



- Using the fact, the projection matrix can be completed.

$$z' = -m_3 - \frac{m_4}{z} \rightarrow \begin{matrix} -1 = -m_3 + \frac{m_4}{f} \\ 0 = -m_3 + \frac{m_4}{n} \end{matrix} \rightarrow \begin{matrix} m_3 = \frac{f}{f-n} \\ m_4 = \frac{nf}{f-n} \end{matrix} \rightarrow \begin{pmatrix} \frac{cot(\frac{fovy}{2})}{aspect} & 0 & 0 & 0 \\ 0 & cot(\frac{fovy}{2}) & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$